Licenciatura Engenharia Informática e Multimédia

Modelação e Programação – MoP

Relatório Trabalho Prático 4 Parte A

Docente André Gomes

Trabalho realizado por:

Fábio Dias, nº 42921

Índice

Índice de Figuras

1. Descrição do Projeto

Para este trabalho prático decidi fazer um jogo onde existe uma personagem que possui três atributos e seis vagas para equipamento. A personagem possuirá uma arma básica no início do jogo e, à medida que se aventura, derrotará inimigos com características geradas aleatoriamente e, consequentemente, receberá novos equipamentos.

O jogador também tem de gerir o seu inventário pois tem vagas limitadas. Desta forma, o jogador decide se quer guardar itens, equipá-los ou desequipá-los ou descartá-los do inventário.

O objetivo do jogo é o jogador chegar o mais longe possível até, eventualmente, perder todos os seus pontos de vida. Assim, volta ao menu onde pode mudar os seus itens por itens melhores que arranjou nas suas aventuras, melhorando os seus atributos, e voltar a aventurar-se.

Dado que podemos ter várias personagens e podemos continuar o progresso anterior, é necessário um sistema de gravação. Este será feito por um documento XML que guarda o nome da personagem, os itens equipados e os itens no inventário.

2. Descrição das Classes

Dado que tanto a nossa personagem como os nossos inimigos partilham atributos e os métodos correspondentes, foi criado uma classe abstrata “Character” e extendem duas classes desta: “Player” e “Enemy”. De forma a adicionar alguma variedade aos inimigos, foi declarada uma *String* que possuirá a sua raça que pode variar entre diversos tipos de criaturas míticas. Quanto à classe Player, esta possuirá dois array de Itens, um que será o inventário e outro que são os equipados. Foram criados diversos métodos de forma a gerir ambos os arrays.

Como já mencionado, também foi criada uma classe Item que servirá para qualquer tipo de item que o jogador pode possuir. Possui atributos, dependendo da sua raridade e que tipo de item é.

Foi desenvolvido uma classe Item Generator que, como o nome indica, vai gerar um novo item sempre que o jogador derrota um inimigo. O número de inimigos derrotados até ao momento terá influência quanto à raridade do objecto. Quanto mais raro, melhor serão os seus atributos.

Também mencionado o sistema de gravação, foi criado uma classe estática *Save System* que cria o ficheiro, caso este não exista, grava o progresso actual, assim como carrega o progresso guardado previamente noutra personagem.

Por fim, temos o aclamado “*game loop”* onde existe o método main e onde ocorrem todas as interacões com o jogo. Vamos ter diversos menus e, para identificar em qual nos encontramos, recorri a um *enum GameState*.

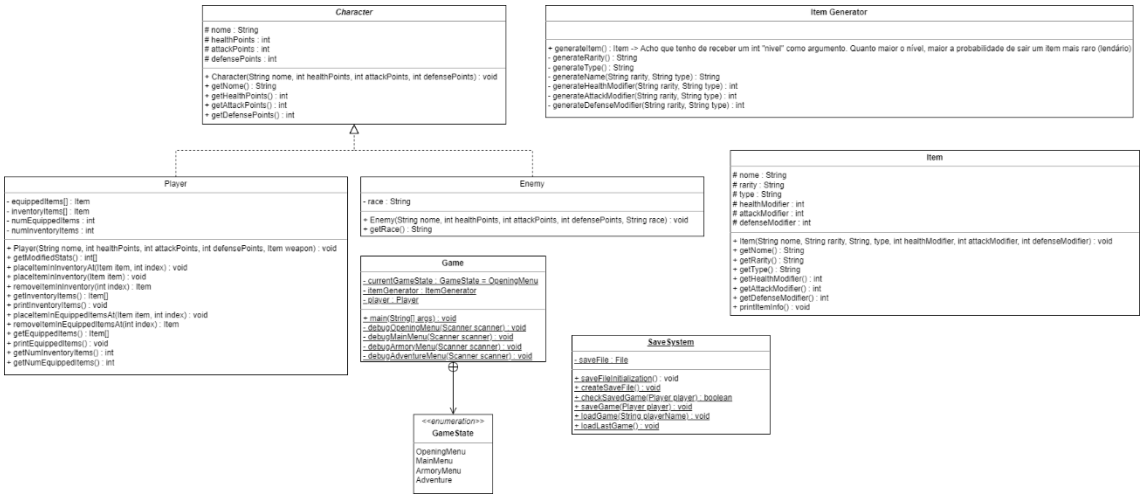
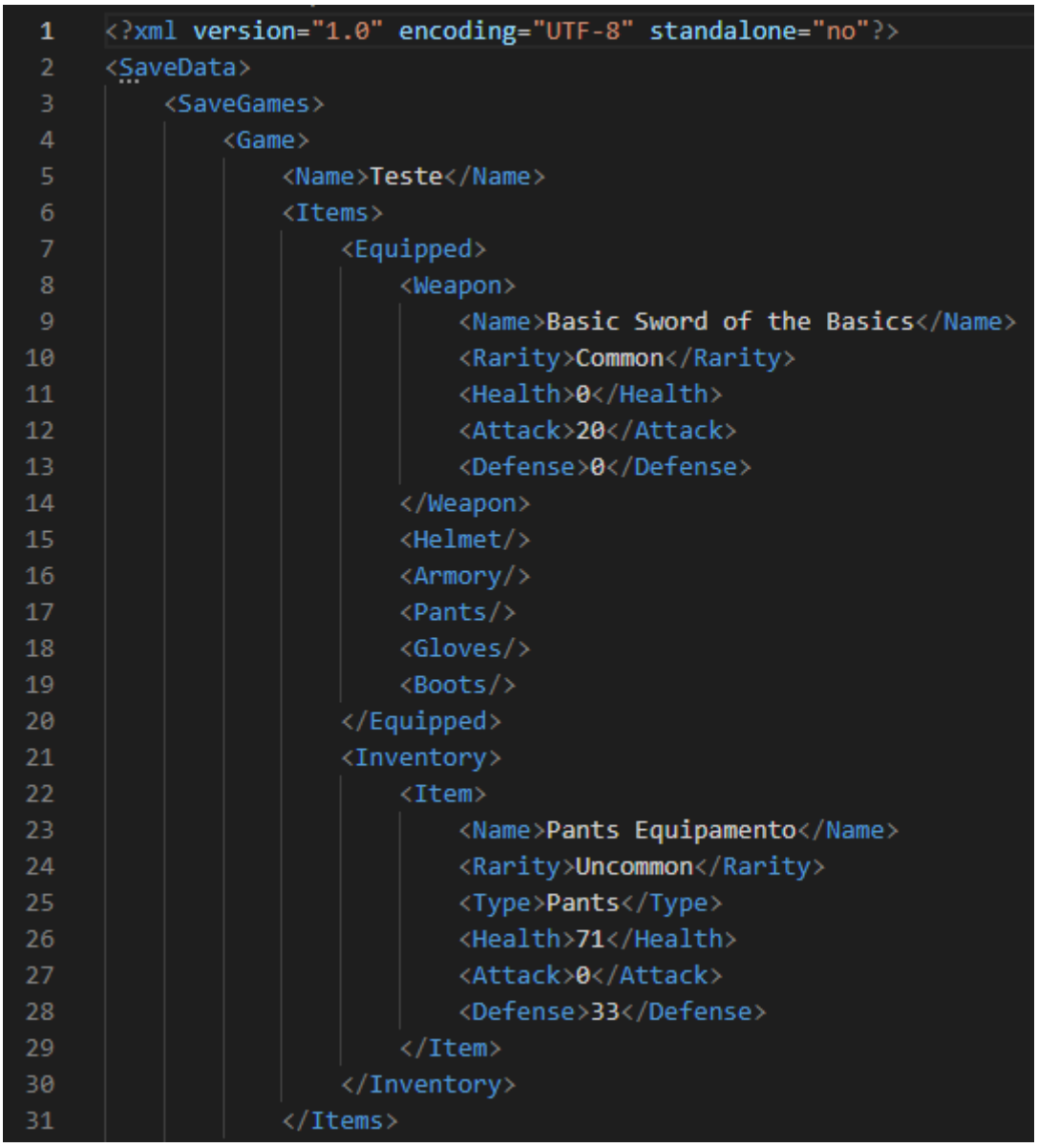

Character
nome : String
healthPoints : int
attackPoints : int
defensePoints : int
+ Character(String nome, int healthPoints, int attackPoints, int defensePoints) : void
+ getNome() : String
+ getHealthPoints() : int
+ getAttackPoints() : int
+ getDefensePoints() : int
Item Generator
+ generateItem() : Item -> Acho que tenho de receber um int "nivel" como argumento. Quanto maior o nível, maior a probabilidade de sair um item mais raro (lendário)
- generateRarity() : String
- generateType() : String
- generateName(String rarity, String type) : String
- generateHealthModifier(String rarity, String type) : int
- generateAttackModifier(String rarity, String type) : int
- generateDefenseModifier(String rarity, String type) : int
Player
- equippedItems[] : Item
- inventoryItems[] : Item
- numEquippedItems : int
- numInventoryItems : int
+ Player(String nome, int healthPoints, int attackPoints, int defensePoints, Item weapon) : void
+ getModifiedStats() : int[]
+ placeItemInInventoryAt(Item item, int index) : void
+ placeItemInInventory(Item item) : void
+ removeItemInInventory(int index) : Item
+ getInventoryItems() : Item[]
+ printInventoryItems() : void
+ placeItemInEquippedItemsAt(Item item, int index) : void
+ removeItemInEquippedItemsAt(int index) : Item
+ getEquippedItems() : Item[]
+ printEquippedItems() : void
+ getNumInventoryItems() : int
+ getNumEquippedItems() : int
Enemy
- race : String
+ Enemy(String nome, int healthPoints, int attackPoints, int defensePoints, String race) : void
+ getRace() : String
Item
nome : String
rarity : String
type : String
healthModifier : int
attackModifier : int
defenseModifier : int
+ Item(String nome, String rarity, String, type, int healthModifier, int attackModifier, int defenseModifier) : void
+ getNome() : String
+ getRarity() : String
+ getType() : String
+ getHealthModifier() : int
+ getAttackModifier() : int
+ getDefenseModifier() : int
+ printItemInfo() : void
Game
- currentGameState : GameState = OpeningMenu
- itemGenerator : ItemGenerator
- player : Player
+ main(String[] args) : void
- debugOpeningMenu(Scanner scanner) : void
- debugMainMenu(Scanner scanner) : void
- debugArmoryMenu(Scanner scanner) : void
- debugAdventureMenu(Scanner scanner) : void
SaveSystem
- saveFile : File
+ saveFileInitialization() : void
+ createSaveFile() : void
+ checkSavedGame(Player player) : boolean
+ saveGame(Player player) : void
+ loadGame(String playerName) : void
+ loadLastGame() : void
«enumeration»
GameState
OpeningMenu
MainMenu
ArmoryMenu
Adventure

Figura 1 - UML

3. Ficheiro de Gravação

Para o ficheiro de gravação, foi necessária a criação de um documento de XML. A organização do mesmo é a partir de um elemento base “SaveData” que possui dois elementos “SaveGames” e “LastSavedGame”. Esta primeira guarda o progresso do jogo a partir da criação de um elemento “Game”, que guarda o nome da personagem no elemento “Name”, e os itens que o jogador possui pelo elemento “Items”. Este é dividido em dois: “Equipped” que cria seis elementos filho que correspondem às vagas dos items que estão equipados actualmente e outro “Inventory” que tem todos os items guardados no inventário.

O elemento LastSavedGame possui o nome da personagem cujo progresso foi guardado da última vez como referência.



```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<SaveData>
    <SaveGames>
        <Game>
            <Name>Teste</Name>
            <Items>
                <Equipped>
                    <Weapon>
                        <Name>Basic Sword of the Basics</Name>
                        <Rarity>Common</Rarity>
                        <Health>0</Health>
                        <Attack>20</Attack>
                        <Defense>0</Defense>
                    </Weapon>
                    <Helmet/>
                    <Armory/>
                    <Pants/>
                    <Gloves/>
                    <Boots/>
                </Equipped>
                <Inventory>
                    <Item>
                        <Name>Pants Equipamento</Name>
                        <Rarity>Uncommon</Rarity>
                        <Type>Pants</Type>
                        <Health>71</Health>
                        <Attack>0</Attack>
                        <Defense>33</Defense>
                    </Item>
                </Inventory>
            </Items>
```

Figura 2 - Ficheiro de Gravação XML

4. Notas

Apenas para testes, o combate não foi implementado e, em vez disso, é apenas gerado um item aleatório e adicionado ao array do inventário, caso seja possível.

O ItemGenerator ainda não implementa nomes variados. Ainda estou a pensar em nomes aleatórios para os items.

Também não foi implementado os métodos de Load Game e Load Last Game do Save System.

Para este jogo foram pensadas diversas mecânicas que tornariam o jogo mais dinâmico. Uma delas era termos diversos ataques que poderiam ser desbloqueados por certos tipos de equipamento. Neste caso atual, temos apenas um ataque mas, com esta mecânica, conseguiríamos ter até quatro ataques distintos pois os itens poderiam fornecer aos jogador esses mesmos.

Outra ideia pensada foi termos efeitos. Caso estes diversos ataques existissem, seria possível termos efeitos que favoreciam o jogador e desfavoreciam o inimigo, assim como o inimigo podia ter efeitos que o favoreciam e que desfavoreciam o jogador. Estes podiam ser “Stun” onde a personagem perderia o turno, “Burn” onde a personagem perdia uma pequena porção de vida durante alguns turnos. “Morale Boost” que aumentaria o dano e a resistência da personagem.